Đã bắt đầu vào lúc	Thứ sáu, 28 Tháng mười 2022, 9:22 PM
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Thứ sáu, 28 Tháng mười 2022, 9:26 PM
Thời gian thực hiện	4 phút 4 giây
Điểm	7,00/7,00
Điểm	**10,00** của 10,00 (**100**%)

Câu hỏi 1

Chính xác

Điểm 1,00 của 1,00

Given a Binary tree, the task is to count the number of nodes with two children

```
#include<iostream>
#include<string>
using namespace std;

template<class K, class V>
class BinaryTree
{
public:
    class Node;

private:
    Node *root;

public:
    BinaryTree() : root(nullptr) {}
    ~BinaryTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }

    class Node
    {
    private:
        K key;
        V value;
        Node *pLeft, *pRight;
        friend class BinaryTree<K, V>;

    public:
        Node(K key, V value) : key(key), value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };

    void addNode(string posFromRoot, K key, V value)
    {
        if(posFromRoot == "")
        {
            this->root = new Node(key, value);
            return;
        }

        Node* walker = this->root;
        int l = posFromRoot.length();
        for (int i = 0; i < l-1; i++)
        {
            if (!walker)
                return;
            if (posFromRoot[i] == 'L')
                walker = walker->pLeft;
            if (posFromRoot[i] == 'R')
                walker = walker->pRight;
        }
        if(posFromRoot[l-1] == 'L')
            walker->pLeft = new Node(key, value);
        if(posFromRoot[l-1] == 'R')
            walker->pRight = new Node(key, value);
    }

    // STUDENT ANSWER BEGIN
    // STUDENT ANSWER END
};
```


You can define other functions to help you.

For example:

Test	Result

BinaryTree<int, int> binaryTree;
binaryTree.addNode("",2, 4); // Add to root
binaryTree.addNode("L",3, 6); // Add to root's left node
binaryTree.addNode("R",5, 9); // Add to root's right node
cout << binaryTree.countTwoChildrenNode();
	1

BinaryTree<int, int> binaryTree;
binaryTree.addNode("",2, 4);
binaryTree.addNode("L",3, 6);
binaryTree.addNode("R",5, 9);
binaryTree.addNode("LL",4, 10);
binaryTree.addNode("LR",6, 2);
cout << binaryTree.countTwoChildrenNode();
	2

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
// STUDENT ANSWER BEGIN
// You can define other functions here to help you.
int recursionCount(int& count, Node* tmp){
    if(tmp->pLeft != nullptr && tmp->pRight != nullptr){
        count++;
        recursionCount(count,tmp->pLeft);
        recursionCount(count,tmp->pRight);
        return count;
    }
    if(tmp->pLeft != nullptr)
    {
        recursionCount(count,tmp->pLeft);
        return count;
    }
    if(tmp->pRight != nullptr) {
        recursionCount(count,tmp->pRight);
        return count;
    }
    return count;
}
int countTwoChildrenNode()
{
    //Node* tmp = root;
    if(!root) return 0;
    int count = 0;
    recursionCount(count,root);
    return count;
}
// STUDENT ANSWER END
```

	Test	Expected	Got	

✔	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("",2, 4); // Add to root`
	`binaryTree.addNode("L",3, 6); // Add to root's left node`
	`binaryTree.addNode("R",5, 9); // Add to root's right node`
	`cout << binaryTree.countTwoChildrenNode();`
		1	1	✔

✔	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("",2, 4);`
	`binaryTree.addNode("L",3, 6);`
	`binaryTree.addNode("R",5, 9);`
	`binaryTree.addNode("LL",4, 10);`
	`binaryTree.addNode("LR",6, 2);`
	`cout << binaryTree.countTwoChildrenNode();`
		2	2	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 2

Chính xác

Điểm 1,00 của 1,00

Given class **BinaryTree**, you need to finish methods **getHeight()**, **preOrder()**, **inOrder()**, **postOrder()**.

```
#include <iostream>
#include <string>
#include <algorithm>
#include <sstream>
using namespace std;

template<class K, class V>
class BinaryTree
{
public:
    class Node;
private:
    Node* root;
public:
    BinaryTree() : root(nullptr) {}
    ~BinaryTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }
    class Node
    {
    private:
        K key;
        V value;
        Node* pLeft, * pRight;
        friend class BinaryTree<K, V>;
    public:
        Node(K key, V value) : key(key), value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
    void addNode(string posFromRoot, K key, V value)
    {
        if (posFromRoot == "")
        {
            this->root = new Node(key, value);
            return;
        }
        Node* walker = this->root;
        int l = posFromRoot.length();
        for (int i = 0; i < l - 1; i++)
        {
            if (!walker)
                return;
            if (posFromRoot[i] == 'L')
                walker = walker->pLeft;
            if (posFromRoot[i] == 'R')
                walker = walker->pRight;
        }
        if (posFromRoot[l - 1] == 'L')
            walker->pLeft = new Node(key, value);
        if (posFromRoot[l - 1] == 'R')
            walker->pRight = new Node(key, value);
    }
    // STUDENT ANSWER BEGIN

    // STUDENT ANSWER END
};
```

For example:

Test	Result

`BinaryTree<int, int> binaryTree;`
`binaryTree.addNode("", 2, 4); // Add to root`
`binaryTree.addNode("L", 3, 6); // Add to root's left node`
`binaryTree.addNode("R", 5, 9); // Add to root's right node`
 	
`cout << binaryTree.getHeight() << endl;`
`cout << binaryTree.preOrder() << endl;`
`cout << binaryTree.inOrder() << endl;`
`cout << binaryTree.postOrder() << endl;`
	2
	4 6 9
	6 4 9
	6 9 4

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
// STUDENT ANSWER BEGIN
// You can define other functions here to help you.
int heightRecursion(Node* root, int currCount){
    if(root == nullptr) return currCount-1;
    int h1 = heightRecursion(root->pLeft,currCount+1);
    int h2 = heightRecursion(root->pRight,currCount+1);
    int max = (h1 < h2)? h2 : h1;
    return max;
    /*if(tmp->pLeft != nullptr && tmp->pRight != nullptr){
        int h1 = heightRecursion(tmp->pLeft,currCount+1);
        int h2 = heightRecursion(tmp->pRight,currCount+1);
        int max = (h1 < h2)? h2 : h1;
        return max;
    }
    if(tmp->pLeft != nullptr) return heightRecursion(tmp->pLeft,currCount+1);
    if(tmp->pRight != nullptr) return heightRecursion(tmp->pRight,currCount+1);
    return currCount;*/
}
int getHeight() {
    return heightRecursion(root,1);
}
string preOrderRe(Node* root){
```


	Test	Expected	Got	

✔	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("", 2, 4); // Add to root`
	`binaryTree.addNode("L", 3, 6); // Add to root's left node`
	`binaryTree.addNode("R", 5, 9); // Add to root's right node`
	
	`cout << binaryTree.getHeight() << endl;`
	`cout << binaryTree.preOrder() << endl;`
	`cout << binaryTree.inOrder() << endl;`
	`cout << binaryTree.postOrder() << endl;`
		2
		4 6 9
		6 4 9
		6 9 4
			2
			4 6 9
			6 4 9
			6 9 4
				✔

✔	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("", 2, 4);`
	
	`cout << binaryTree.getHeight() << endl;`
	`cout << binaryTree.preOrder() << endl;`
	`cout << binaryTree.inOrder() << endl;`
	`cout << binaryTree.postOrder() << endl;`
		1
		4
		4
		4
			1
			4
			4
			4
				✔

	Test	Expected	Got	

✔	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("", 2, 4);
	binaryTree.addNode("L", 3, 6);
	binaryTree.addNode("R", 5, 9);
	binaryTree.addNode("LL", 4, 10);
	binaryTree.addNode("LR", 6, 2);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		3
		4 6 10 2 9
		10 6 2 4 9
		10 2 6 9 4
			3
			4 6 10 2 9
			10 6 2 4 9
			10 2 6 9 4
				✔

✔	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("", 2, 4);
	binaryTree.addNode("L", 3, 6);
	binaryTree.addNode("R", 5, 9);
	binaryTree.addNode("LL", 4, 10);
	binaryTree.addNode("RL", 6, 2);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		3
		4 6 10 9 2
		10 6 4 2 9
		10 6 2 9 4
			3
			4 6 10 9 2
			10 6 4 2 9
			10 6 2 9 4
				✔

✔	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LLL",6, 2);
	binaryTree.addNode("LLLR",7, 7);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		5
		4 6 10 2 7 9
		2 7 10 6 4 9
		7 2 10 6 9 4
			5
			4 6 10 2 7 9
			2 7 10 6 4 9
			7 2 10 6 9 4
				✔

✔	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LLL",6, 2);
	binaryTree.addNode("LLLR",7, 7);
	binaryTree.addNode("RR",8, 30);
	binaryTree.addNode("RL",9, 307);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		5
		4 6 10 2 7 9 307 30
		2 7 10 6 4 307 9 30
		7 2 10 6 307 30 9 4
			5
			4 6 10 2 7 9 307 30
			2 7 10 6 4 307 9 30
			7 2 10 6 307 30 9 4
				✔

	Test	Expected	Got	

✔	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LR",6, -3);
	binaryTree.addNode("LLL",7, 2);
	binaryTree.addNode("LLLR",8, 7);
	binaryTree.addNode("RR",9, 30);
	binaryTree.addNode("RL",10, 307);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		5
		4 6 10 2 7 -3 9 307 30
		2 7 10 6 -3 4 307 9 30
		7 2 10 -3 6 307 30 9 4
			5
			4 6 10 2 7 -3 9 307 30
			2 7 10 6 -3 4 307 9 30
			7 2 10 -3 6 307 30 9 4
				✔

✔	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LR",6, -3);
	binaryTree.addNode("LLL",7, 2);
	binaryTree.addNode("LLLR",8, 7);
	binaryTree.addNode("RR",9, 30);
	binaryTree.addNode("RL",10, 307);
	binaryTree.addNode("RLL",11, 2000);
	binaryTree.addNode("RLR",12, 2000);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		5
		4 6 10 2 7 -3 9 307 2000
		2000 30
		2 7 10 6 -3 4 2000 307 2000
		9 30
		7 2 10 -3 6 2000 2000 307 30
		9 4
			5
			4 6 10 2 7 -3 9 307 2000
			2000 30
			2 7 10 6 -3 4 2000 307 2000
			9 30
			7 2 10 -3 6 2000 2000 307
			30 9 4
				✔

✔	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LR",6, -3);
	binaryTree.addNode("LLL",7, 2);
	binaryTree.addNode("LLLR",8, 7);
	binaryTree.addNode("RR",9, 30);
	binaryTree.addNode("RL",10, 307);
	binaryTree.addNode("RLL",11, 2000);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		5
		4 6 10 2 7 -3 9 307 2000 30
		2 7 10 6 -3 4 2000 307 9 30
		7 2 10 -3 6 2000 307 30 9 4
			5
			4 6 10 2 7 -3 9 307 2000 30
			2 7 10 6 -3 4 2000 307 9 30
			7 2 10 -3 6 2000 307 30 9 4
				✔

	Test	Expected	Got	

✔	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("",2, 4);`
	`binaryTree.addNode("L",3, 6);`
	`binaryTree.addNode("R",5, 9);`
	`binaryTree.addNode("LL",4, 10);`
	`binaryTree.addNode("LR",6, -3);`
	`binaryTree.addNode("LLL",7, 2);`
	`binaryTree.addNode("LLLR",8, 7);`
	`binaryTree.addNode("RR",9, 30);`
	`binaryTree.addNode("RL",10, 307);`
	`binaryTree.addNode("RLL",11, 2000);`
	`binaryTree.addNode("RLLL",11, 2000);`
	
	`cout << binaryTree.getHeight() << endl;`
	`cout << binaryTree.preOrder() << endl;`
	`cout << binaryTree.inOrder() << endl;`
	`cout << binaryTree.postOrder() << endl;`
		5
		4 6 10 2 7 -3 9 307 2000 2000 30
		2 7 10 6 -3 4 2000 2000 307 9 30
		7 2 10 -3 6 2000 2000 307 30 9 4
			5
			4 6 10 2 7 -3 9 307 2000 2000 30
			2 7 10 6 -3 4 2000 2000 307 9 30
			7 2 10 -3 6 2000 2000 307 30 9 4
				✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 3

Chính xác

Điểm 1,00 của 1,00

Given a Binary tree, the task is to calculate the sum of leaf nodes. (Leaf nodes are nodes which have no children)

```
#include<iostream>
#include<string>
using namespace std;

template<class K, class V>
class BinaryTree
{
public:
    class Node;
private:
    Node *root;

public:
    BinaryTree() : root(nullptr) {}
    ~BinaryTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }

    class Node
    {
    private:
        K key;
        V value;
        Node *pLeft, *pRight;
        friend class BinaryTree<K, V>;

    public:
        Node(K key, V value) : key(key), value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };

    void addNode(string posFromRoot, K key, V value)
    {
        if(posFromRoot == "")
        {
            this->root = new Node(key, value);
            return;
        }

        Node* walker = this->root;
        int l = posFromRoot.length();
        for (int i = 0; i < l-1; i++)
        {
            if (!walker)
                return;
            if (posFromRoot[i] == 'L')
                walker = walker->pLeft;
            if (posFromRoot[i] == 'R')
                walker = walker->pRight;
        }
        if(posFromRoot[l-1] == 'L')
            walker->pLeft = new Node(key, value);
        if(posFromRoot[l-1] == 'R')
            walker->pRight = new Node(key, value);
    }
    //Helping functions
    int sumOfLeafs(){
        //TODO
     }
};
```


You can write other functions to achieve this task.

For example:

Test	Result

BinaryTree<int, int> binaryTree;
binaryTree.addNode("", 2, 4);
cout << binaryTree.sumOfLeafs();
	4

BinaryTree<int, int> binaryTree;
binaryTree.addNode("", 2, 4);
binaryTree.addNode("L", 3, 6);
binaryTree.addNode("R", 5, 9);
cout << binaryTree.sumOfLeafs();
	15

Answer: (penalty regime: 0 %)

Reset answer

```
//Helping functions
int sumOfLeafs(Node*root){
    if(root == nullptr) return 0;
    if(root->pLeft == nullptr && root->pRight == nullptr) return root->value;
    return sumOfLeafs(root->pLeft) + sumOfLeafs(root->pRight);
}
int sumOfLeafs(){
    //int sum = 0;
    return sumOfLeafs(root);

}
/*
            1
        2           3
    4       6   7       8
5

*/
```

	Test	Expected	Got	

✔	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("", 2, 4);`
	`cout << binaryTree.sumOfLeafs();`
		4	4	✔

✔	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("", 2, 4);`
	`binaryTree.addNode("L", 3, 6);`
	`binaryTree.addNode("R", 5, 9);`
	`cout << binaryTree.sumOfLeafs();`
		15	15	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 4

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary tree, described on the following:

```
class BTNode {
    public:
        int val;
        BTNode *left;
        BTNode *right;
        BTNode() {
            this->left = this->right = NULL;
        }
        BTNode(int val) {
            this->val = val;
            this->left = this->right = NULL;
        }
        BTNode(int val, BTNode*& left, BTNode*& right) {
            this->val = val;
            this->left = left;
            this->right = right;
        }
};
```

Where `val` is the value of node (integer, in segment `[0,9]`), `left` and `right` are the pointers to the left node and right node of it, respectively.

Request: Implement function:

```
int sumDigitPath(BTNode* root);
```

Where `root` is the root node of given binary tree (this tree has between 2 and 100000 elements). This function returns the sum of all **digit path** numbers of this binary tree (the result may be large, so you must use `mod 27022001` before returning).

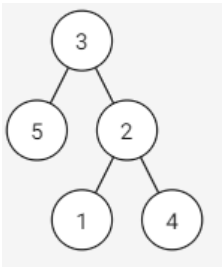
More information:

- A path is called as **digit path** if it is a path from the root node to the leaf node of the binary tree.

- Each **digit path** represents a number in order, each node's `val` of this path is a digit of this number, while root's `val` is the first digit.

Example:

Given a binary tree in the following:



All of the **digit paths** are `3-5`, `3-2-1`, `3-2-4`; and the number reprensted by them are `35`, `321`, `324`, respectively. The sum of them (after `mod 27022001`) is `680`.

Note: In this exercise, the libraries `iostream`, `queue`, `stack`, `utility` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result
int arr[] = {-1,0,0,2,2}; int value[] = {3,5,2,1,4}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << sumDigitPath(root);	680
int arr[] = {-1,0,0}; int value[] = {1,2,3}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << sumDigitPath(root);	25

Answer: (penalty regime: 0 %)

Reset answer

```
int stToInt(stack<int> st){
    vector<int> q;
    while(!st.empty()){
        q.insert(q.begin(),st.top());
        st.pop();
    }
    int ans = 0;
    while(!q.empty()){
        int tmp = q[0];
        q.erase(q.begin());
        ans = ((ans*10) +tmp)%27022001  ;
    }
    return ans;
}
int sumDigitPathRec(BTNode* root,stack<int>& st){
    if(root == nullptr){
        return 0;
    }
    if(root->left == nullptr && root->right == nullptr){
        st.push(root->val);
        int ans = stToInt(st);
        st.pop();
        return ans;
    }
    st.push(root->val);
    int ans =  (sumDigitPathRec(root->left,st)+sumDigitPathRec(root->right,st))%27022001;
    st.pop();
    return ans;
}
int sumDigitPath(BTNode* root) {
    stack<int> st;
    return sumDigitPathRec(root,st);
}
```

	Test	Expected	Got	

✔	`int arr[] = {-1,0,0,2,2};`
	`int value[] = {3,5,2,1,4};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
	`cout << sumDigitPath(root);`
		680	680	✔

✔	`int arr[] = {-1,0,0};`
	`int value[] = {1,2,3};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
	`cout << sumDigitPath(root);`
		25	25	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 5

Chính xác

Điểm 1,00 của 1,00

Given a Binary tree, the task is to traverse all the nodes of the tree using Breadth First Search algorithm and print the order of visited nodes (has no blank space at the end)

```
#include<iostream>
#include<string>
#include<queue>
using namespace std;

template<class K, class V>
class BinaryTree
{
public:
    class Node;

private:
    Node *root;

public:
    BinaryTree() : root(nullptr) {}
    ~BinaryTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }

    class Node
    {
    private:
        K key;
        V value;
        Node *pLeft, *pRight;
        friend class BinaryTree<K, V>;

    public:
        Node(K key, V value) : key(key), value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };

    void addNode(string posFromRoot, K key, V value)
    {
        if(posFromRoot == "")
        {
            this->root = new Node(key, value);
            return;
        }

        Node* walker = this->root;
        int l = posFromRoot.length();
        for (int i = 0; i < l-1; i++)
        {
            if (!walker)
                return;
            if (posFromRoot[i] == 'L')
                walker = walker->pLeft;
            if (posFromRoot[i] == 'R')
                walker = walker->pRight;
        }
        if(posFromRoot[l-1] == 'L')
            walker->pLeft = new Node(key, value);
        if(posFromRoot[l-1] == 'R')
            walker->pRight = new Node(key, value);
    }

    // STUDENT ANSWER BEGIN
    // STUDENT ANSWER END
};
```

You can define other functions to help you.

For example:

Test	Result

`BinaryTree<int, int> binaryTree;`
`binaryTree.addNode("",2, 4); // Add to root`
`binaryTree.addNode("L",3, 6); // Add to root's left node`
`binaryTree.addNode("R",5, 9); // Add to root's right node`
`binaryTree.BFS();`
	4 6 9

Answer: (penalty regime: 0 %)

Reset answer

```
// STUDENT ANSWER BEGIN
// You can define other functions here to help you.

void BFS()
{
    if(root == nullptr) return;
    Node* tmp = root;
    cout<< tmp->value;
    queue<Node*> q;
    q.push(tmp->pLeft);
    q.push(tmp->pRight);
    while(!q.empty()){
        Node* curr = q.front();
        q.pop();
        if(curr != nullptr){
            cout<< " " << curr->value;
            q.push(curr->pLeft);
            q.push(curr->pRight);
        }
    }

}
// STUDENT ANSWER END
```

	Test	Expected	Got	

✔	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("",2, 4); // Add to root`
	`binaryTree.addNode("L",3, 6); // Add to root's left node`
	`binaryTree.addNode("R",5, 9); // Add to root's right node`
	`binaryTree.BFS();`
		4 6 9	4 6 9	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 6

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary tree, described on the following:

```
class BTNode {
    public:
        int val;
        BTNode *left;
        BTNode *right;
        BTNode() {
            this->left = this->right = NULL;
        }
        BTNode(int val) {
            this->val = val;
            this->left = this->right = NULL;
        }
        BTNode(int val, BTNode*& left, BTNode*& right) {
            this->val = val;
            this->left = left;
            this->right = right;
        }
};
```

Where `val` is the value of node (non-negative integer), `left` and `right` are the pointers to the left node and right node of it, respectively.

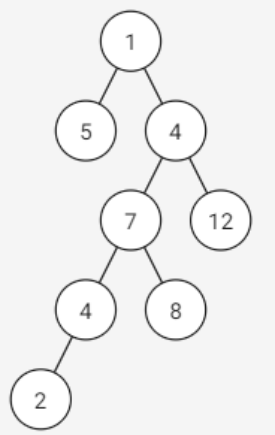
Request: Implement function:

```
int longestPathSum(BTNode* root);
```

Where `root` is the root node of given binary tree (this tree has between 1 and 100000 elements). This function returns the sum of the largest path from the root node to a leaf node. If there are more than one equally long paths, return the larger sum.

Example:

Given a binary tree in the following:



The longest path from the root node to the leaf node is `1-4-7-4-2`, so return the sum of this path, is `18`.

Note: In this exercise, the libraries `iostream`, `utility`, `queue`, `stack` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result
int arr[] = {-1,0,0,2,2,3,3,5}; int value[] = {1,5,4,7,12,4,8,2}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << longestPathSum(root);	18
int arr[] = {-1,0,1,0,1,4,5,3,7,3}; int value[] = {6,12,23,20,20,20,3,9,13,15}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << longestPathSum(root);	61

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
int sumSt(stack<int> st){
    int ans = 0;
    while(!st.empty()){
        ans += st.top();
        st.pop();
    }
    return ans;
}

stack<int> longestPath(BTNode* root, stack<int> st){
    if(root == nullptr){
        return st;
    }
    st.push(root->val);
    stack<int> st1 = longestPath(root->left,st);
    stack<int> st2 = longestPath(root->right,st);
    if(st1.size() > st2.size()) st = st1;
    else if(st1.size() < st2.size()) st = st2;
    else {
        int sum1 = sumSt(st1);
        int sum2 = sumSt(st2);
        st = (sum1 > sum2)? st1 : st2;
    }
    return st;
}

int longestPathSum(BTNode* root) {
    stack<int> st;
    st = longestPath(root,st);
    int ans = sumSt(st);
    return ans;
}
```

	Test	Expected	Got	

✔	`int arr[] = {-1,0,0,2,2,3,3,5};`
	`int value[] = {1,5,4,7,12,4,8,2};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
	`cout << longestPathSum(root);`
		18	18	✔

✔	`int arr[] = {-1,0,1,0,1,4,5,3,7,3};`
	`int value[] = {6,12,23,20,20,20,3,9,13,15};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
	`cout << longestPathSum(root);`
		61	61	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 7

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary tree, described on the following:

```
class BTNode {
    public:
        int val;
        BTNode *left;
        BTNode *right;
        BTNode() {
            this->left = this->right = NULL;
        }
        BTNode(int val) {
            this->val = val;
            this->left = this->right = NULL;
        }
        BTNode(int val, BTNode*& left, BTNode*& right) {
            this->val = val;
            this->left = left;
            this->right = right;
        }
};
```

Where `val` is the value of node (non-negative integer), `left` and `right` are the pointers to the left node and right node of it, respectively.

Request: Implement function:

```
int lowestAncestor(BTNode* root, int a, int b);
```

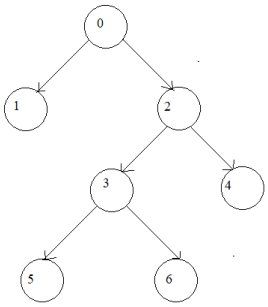
Where `root` is the root node of given binary tree (this tree has between 2 and 100000 elements). This function returns the **lowest ancestor** node's `val` of node `a` and node `b` in this binary tree (assume a and b always exist in the given binary tree).

More information:

- A node is called as the **lowest ancestor** node of node `a` and node `b` if node `a` and node `b` are its descendants.
- A node is also the descendant of itself.
- On the given binary tree, each node's `val` is distinguish from the others' `val`

Example:

Given a binary tree in the following:



- The **lowest ancestor** of node `4` and node `5` is node `2`.

Note: In this exercise, the libraries `iostream, stack, queue, utility` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result
int arr[] = {-1,0,0,2,2,3,3}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL); cout << lowestAncestor(root, 4, 5);	2
int arr[] = {-1,0,1,1,0,4,4,2,5,6}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL); cout << lowestAncestor(root, 4, 9);	4

Answer: (penalty regime: 0 %)

Reset answer

```
queue<int> searchDFS(BTNode* root, int x,queue<int> q){
    if(root == nullptr){
        queue<int> nullq;
        q = nullq;
        return q;
    }
    if(root->val == x){
        q.push(x);
        return q;
    }
    q.push(root->val);
    queue<int> q1 = searchDFS(root->left,x,q);
    queue<int> q2 = searchDFS(root->right,x,q);
    if(!q1.empty()) q = q1;
    else q = q2;
    return q;
}


int lowestAncestor(BTNode* root, int a, int b) {
    queue<int> qa;
    queue<int> qb;
    qa = searchDFS(root,a,qa);
    qb = searchDFS(root,b,qb);
    bool isDone = false;
    int ans = root->val;
    while(!isDone){
        if(qa.front() == qb.front()) {
            ans = qa.front();
            qa.pop();
            qb.pop();
        }
        else {
            isDone = true;
        }
    }
    return ans;

}
```

	Test	Expected	Got	

✔	`int arr[] = {-1,0,0,2,2,3,3};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL);`
	`cout << lowestAncestor(root, 4, 5);`
		2	2	✔

✔	`int arr[] = {-1,0,1,1,0,4,4,2,5,6};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL);`
	`cout << lowestAncestor(root, 4, 9);`
		4	4	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.